

METHOD AND APPARATUS FOR MULTICAST SUPPORT

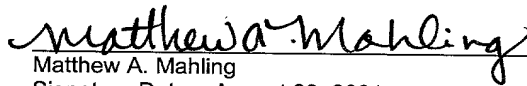
INVENTORS

Shean-Guang Chang
Stephan Zachweija

**CERTIFICATE OF MAILING BY "EXPRESS MAIL"
UNDER 37 C.F.R. § 1.10**

"Express Mail" mailing label number: **EL 622 696 814 US**
Date of Mailing: August 29, 2001

I hereby certify that this correspondence is being deposited with the United States Postal Service, utilizing the "Express Mail Post Office to Addressee" service addressed to **Box PATENT APPLICATION, Commissioner for Patents, Washington, D.C. 20231** and mailed on the above Date of Mailing with the above "Express Mail" mailing label number.



Matthew A. Mahling

Signature Date: August 29, 2001

METHOD AND APPARATUS FOR MULTICAST SUPPORT

INVENTORS:

Shean-Guang Chang
Stephan Zachweija

CLAIM OF PRIORITY

[0001] This application claims priority to Provisional patent application Serial No. 60/305,985, filed July 16, 2001, entitled METHOD AND APPARATUS FOR MULTICAST SUPPORT.

5

COPYRIGHT NOTICE

[0002] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

10

TECHNICAL FIELD

[0003] The invention relates generally to enterprise messaging systems and specifically to a system and a method for concurrently offering multiple qualities of service from a single messaging source.

15

BACKGROUND

[0004] The Java Message Service ("JMS") Application Programming Interface ("API"), part of the Java™ 2 Platform, Enterprise Edition ("J2EE") produced by Sun Microsystems, Inc. of Palo Alto, California, is a software interface that is useful in accessing enterprise messaging systems. JMS provides the ability for applications to send and receive data asynchronously, by defining a common enterprise messaging API supported by several major enterprise messaging products. JMS supports both message queueing and subscription-based messaging. Enterprise Java beans, components well known to those skilled in the computer arts, use the JMS API to send enterprise messages and receive them either synchronously or asynchronously.

[0005] The JMS API does not provide end-to-end synchronous message delivery and notification. Other messaging systems can provide synchronous delivery as a mechanism for implementing reliable applications, or can provide clients with delivery notification if messages are not received. The JMS API, on the other hand, provides only a guaranteed single delivery of messages. The JMS API does not utilize system messages, such as delivery notifications. Any notification must be done at the application level.

[0006] Further, JMS does not offer true multicasting services. JMS offers only a simple "Acknowledge" mode, which allows a user to subscribe to specific "topics," or messages arranged or grouped such as by subject.

JMS does not provide the capability to specify selection criteria or filter out locally-produced messages. Multicast is not supported for queue consumers or durable consumers. Further, multicast itself is based on an unreliable protocol, such that messages can be lost. Using the “one time only” sending of messages with the JMS API would mean that these lost messages could not be recovered. This may be undesirable to some users.

[0007] In other synchronous queuing systems of the prior art, it is typically necessary to have one API for queuing and reliable transport. These systems usually have to continuously de-queue in order to get all the incoming messages. If a user wished to also receive multicast traffic, it would be necessary to use a different API. In such a raw mode, it is necessary to read a socket, then queue, then the socket, then queue, etc. The system would need to be able to handle both at the same time, having to do the multiplexing and time slicing itself. The developer would also have to understand and program for two different APIs. Even if the developer is not doing the raw multicasting at the operating system level, the developer typically wraps the system with customized APIs. If it is desired to receive a different quality of service, it will be necessary to switch APIs altogether.

[0008] It is therefore desirable to develop a messaging system that utilizes the advantages of multicasting, but also offers improved reliability to those users who do not wish to lose messages.

[0009] It is also desirable to develop such a messaging system that functions without the need for separate APIs.

BRIEF SUMMARY

5 [0010] The present invention includes a system and method for providing two qualities of service from a single data stream. A storage space is used to store a first quality of service choice and/or a second quality of service choice for each user of the system. A processor of the system is programmed to direct the data stream, and/or any messages on
10 the data stream, for each user, according to that user's quality of service choice. The system contains multicasting apparatus for receiving the data stream from the processor and multicasting the data stream to each user for whom the first quality of service choice is stored. The system also contains a device, such as a point-to-point device, for receiving the data
15 stream from the processor and ensuring that each user for whom the second quality of service is stored receives the data stream and/or messages.

BRIEF DESCRIPTION OF THE DRAWINGS

20 [0011] Figure 1 is a diagram of a system in accordance with one embodiment of the present invention demonstrating a point-to-point quality of service for each user.

[0012] Figure 2 is a diagram of a system in accordance with one embodiment of the present invention demonstrating a multicast quality of service for each user.

5 **[0013]** Figure 3 is a diagram of a system in accordance with one embodiment of the present invention demonstrating both qualities of service.

[0014] Figure 4 is a diagram of a system in accordance with one embodiment of the present invention demonstrating both qualities of service.

10 **[0015]** Figure 5 is a diagram of a system in accordance with one embodiment of the present invention demonstrating both qualities of service.

[0016] Figure 6 is a flowchart showing steps of a messaging process in accordance with one embodiment of the present invention.

15

DETAILED DESCRIPTION

[0017] The present invention overcomes many of the deficiencies of prior art messaging systems. Messaging systems of the prior art, such as the JMS API, typically establish a source of information to be broadcast.

20 Users of these systems publish information to this source, which is then used for broadcasting. The source and publication in this method are, however, fairly unreliable. It is up to users of these systems to develop

and implement their own retransmission schemes if the users do not wish to miss messages.

[0018] A system in accordance with the present invention can instead send messages to a server that will do a multicast. This server cannot only provide the multicasting service, which is described as unreliable above, but can also offer a reliable service. These services can operate concurrently or sequentially, and can alternate services for each message. Unlike prior art systems, such a system is not configured as either a multicast or a reliable source alone, but offers both services through the same channel.

[0019] In one embodiment, a user can specify the quality of service ("QoS") to be used for messages sent to, or received by, that user. Each user of the system can have associated attributes, one of which is quality of service. For users selecting a higher quality of service, additional information may be provided, such as a message being sent to inform those users of a message being multicast. Some users may configure or enable a source to be multicastable, while other subscribers may want reliability and do not wish to miss any messages. A system in accordance with one embodiment of the present invention is actually servicing both user communities with different qualities of service from a single source. This dual servicing may be completely transparent to users of either QoS.

[0020] **Figure 1** shows a system **100** in accordance with the present invention, in which each user **118, 120, 122, 124** has selected a reliable

quality of service, in this case utilizing a point-to-point protocol. In **Figure 1**, a sender **102** sends a single message along a data stream **104** to a system server **106**. The system server **106** sends the message **108** to the network **112**, which delivers the message **114** to User A **118**, who has
5 selected to receive messages by the reliable quality of service. User A **118** sends a response **116** back to the network **112**, which delivers an acknowledgment of receipt **110** to the system server **106**. The system server **106** also follows these steps for User B **120**, User C **122**, and User D **124**. These steps may be followed for multiple users serially, in parallel,
10 or concurrently, and may be done in any appropriate order among the users.

[0021] **Figure 2** shows the same system **100**, in which each user **118, 120, 122, 124** has selected a faster but less reliable quality of service, in this case utilizing a multicast protocol. In **Figure 2**, the sender **102**
15 sends a single message along the data stream **104** to the system server **106**. The system server **106** sends a single message **126** to the network **112**, which delivers the message **114** to User A **118**, User B **120**, User C **122**, and User D **124**, who have selected to receive messages by the less reliable quality of service. For this quality of service, the users **118, 120,**
20 **122, 124** do not send a response that needs to be received by the system server **106**. Each user may receive the message at approximately the same time, but is not ensured of getting the message.

[0022] In **Figure 3**, the sender **102** again sends a single message along the data stream **104** to the system server **106**. The system server **106** looks to a storage space **142**, such as volatile or runtime memory for temporary storage, to determine which users have selected the reliable quality of service and which users have selected the less reliable quality of service.

[0023] For User A **118** and User C **122**, the system server **106** sends a single message **130** to the network **112**, which delivers the message **136** to User A **118** and User C **122**, who have selected to receive messages by the less reliable quality of service. For this quality of service, User A **118** and User C **122** do not send a response that needs to be received by the system server **106**. User A **118** and User C **122** may receive the message **136** at approximately the same time, but they are not ensured of getting the message.

[0024] The system server **106** also sends the message **132** to the network **112**, which delivers the message **138** to User B **120**, who has selected to receive messages by the reliable quality of service. User B **120** sends a response **140** back to the network **112**, which delivers an acknowledgment of receipt **134** to the system server **106**. The system server **106** also undergoes the reliable process for User D **124**.

[0025] **Figure 4** shows another system **200** in accordance with one embodiment of the present invention. The sender **202** sends a single

message along a data stream **204** to a system processor **206**. The system processor **206** looks to a storage space **230**, such as runtime or volatile memory for temporary storage, to determine which users have selected the reliable quality of service and which users have selected the less reliable quality of service. For User A **214** and User C **218**, the system processor **206** sends a single message **208** to a multicast apparatus **210**, which delivers the message **212** to User A **214** and User C **218**, who have selected to receive messages by the less reliable quality of service. For this quality of service, User A **214** and User C **218** do not send a response. User A **214** and User C **218** may receive the message **212** at approximately the same time, but they are not ensured to get the message **212**.

[0026] The system processor **206** also sends the message **222** to a point-to-point apparatus **224**, which delivers the message **226** to User B **216**, who has selected to receive messages by the reliable quality of service. User B **216** then sends a response **228** back to the point-to-point apparatus **224**. The point-to-point apparatus **224** also undergoes the point-to-point process for User D **220**. When implementing multicast and point-to-point distribution, multicast can be initiated or scheduled before, after, or at the same time as point-to-point. The individual tasks for the two distribution methods can compete for resources and can be executed in any order and/or at the same time.

[0027] **Figure 5** shows the same system **200**, except that this time

User B **216** has selected to receive messages by both qualities of service. This may be appropriate, for example, when users wish to receive messages quickly, such as by the unreliable QoS, but also wish to ensure that they receive all the messages, thereby requesting that messages also
5 be sent by the reliable QoS. For User A **214** , User B **216**, and User C **218**, the system processor **206** sends a single message **208** to a multicast apparatus **210**, which delivers the message **232** to User A **214** , User B **216**, and User C **218**, who have selected to receive messages by the less reliable quality of service. The system processor **206** also sends the
10 message **222** to a point-to-point apparatus **224**, which delivers the message **226** to User B **216** and User D **220**, who have selected to receive messages by the reliable quality of service. User B **216** sends a response **228** back to the point-to-point apparatus **224** for the reliable service only.

15 **[0028]** For both the system of Figure 1 and the system of Figure 4, a process **300** may be used in accordance with the present invention that is shown in **Figure 6**. In the process, a first and/or second quality of service choice is stored for each user of the system, or at least for each user who has selected to receive a message from that sender **302**. A message
20 received on a data stream is directed to each user according to that user's quality of service choice **304**. For users selecting the first quality of service choice, the message is multicast to each such user at approximately the same time **306**. For users selecting the second quality of service, the

messages are sent individually to each such user and it is ensured that each user receives the message **308**.

[0029] One reason why some users may prefer the multicast approach is that there are some penalties are inherent in the fully reliable service. One such penalty involves the diminution of overall system performance. The overall performance of the system may be diminished. The fully reliable service is typically both slower and less scalable than the multicasting system. When multicasting, a message is sent only once no matter how many subscribers exist. Multicast messages can be delivered via a routine such as an onMessage routine. Users can create consumers in their multicast session and register listeners for those consumers. Messages from the consumers may then be serialized across the session just like other asynchronous messages.

[0030] Another reason for preferring the multicast approach lies in the fact that some applications are very sensitive to any delivery time latency. Applications can also be sensitive to the order in which different receivers obtain a message. To get a message later than another receiver can be a big disadvantage, while receiving a message earlier can be a big advantage. When using multicast, these applications and receivers get messages at approximately the same time, such that there is almost no problem with latency and/or ordering issues.

[0031] In the reliable case, it can be necessary to contact, and wait on, each subscriber. It may be impracticable to scale to large numbers

when using the reliable system, as the service may require a handshake to guarantee that all subscribers have received the data. If this approach is used for a sufficiently large number of users, the system may not be able to keep up with a heavy messaging load.

5 **[0032]** A system in accordance with one embodiment of the present invention avoids these problems by offering a QoS choice to the user. A user or server having a message to be broadcast may only need to send each message to the system once. The system can handle both qualities of service for that message. One embodiment segregates the user list into
10 two groups, one for each quality of service. These groups may be maintained in any appropriate storage manner, such as in volatile or runtime memory for temporary storage or in a system database for systems or applications requiring more persistent storage.

15 **[0033]** The system can also do client-side filtering rather than server-side filtering. This can improve the performance of the system by moving some of the functionality from the server to the client. Filtering can be done on the client side such as by JMS, before the message is given to an application. While the filtering is done on the client side, it is JMS and not the application that is in control of filtering. In the case of point-to-point
20 delivery, the filtering can also be done on the server side by JMS. When a message arrives at a client, by either QoS, the client or user can always choose to filter the message out at that time. Various methods of message filtering may be used, such as are known and used in the art.

[0034] As discussed above, the JMS API allows a user to specify the level of handshaking, such as a level using "Acknowledge" mode. In Acknowledge mode, the user can choose to acknowledge all, some, or none of the incoming messages. Acknowledge mode is the switch as far as the user is concerned, and a listener can be used to receive the messages. When the multicast is set to "No Acknowledge" mode, it may not be necessary to actually change the way in which the messages are received. A client is still handed messages one at a time, so nothing typically needs to be done in order to receive message without having to acknowledge.

[0035] In prior art systems, it would be necessary to change the program APIs in order to change the QoS. A customer may have subscribers who prefer multicast and subscribers who prefer more reliable service, but would prefer not to have to create and manage two APIs in order to accommodate both groups.

[0036] In a system in accordance with the present invention, a user can simply set one parameter and receive the messages using the desired QoS. In this system, there is a thread of execution for multicast subscribers. The thread of execution can comprise a program running in the background that listens on a user's behalf. When a message arrives on a multicast stream, the message can be packaged like a normal JMS message and delivered to the listener. Whenever the message arrives, the listener can be invoked, such as through a callback.

[0037] For reliable subscribers, a server can call the client directly.

In such a case, the server can hand the message directly to the client, then package and deliver the message to the listener as well. Both messages can come into the listener. A program, such as a small "grab back" program, may be used to catch the messages and hand them to the clients. In this manner, the messages can look the same to the client.

[0038] A permanent listener can be put on the service for a user.

Whenever a message arrives, it can be queued up for a regular listener that has been set up by the user. A thread of execution can listen continually to the input stream or socket, putting any received message in the queue. In the JMS sense, there is a single-ordered stream of messages coming in, or a "session." A session in the JMS sense is a multicast session created by specifying an "Acknowledge" mode.

[0039] Within a session, a user can express interest in multiple input

streams. If all these streams are multicast streams, there may be only one socket and only one thread that listens on that socket. When a message is received, it can be delivered to the user in a way that the user may differentiate the source of the message. A user can also create multiple sessions, in order to have multiple threads listening. In fact, a user can have a dedicated thread for each topic, if the user so chooses, each thread residing on a dedicated socket.

[0040] If a user has multiple sources of information, the user can choose to hear everything and sort it out later. The user can also choose

to only listen to one or two sources. Each source can be configured to use a separate medium. A user can choose to set up several topics on one socket or "multicast group." If the user is only interested in one topic, it may be necessary to throw away all of the other topics coming at the user.

5 **[0041]** While processing one message, it is possible to miss another message. The system may be able to tell the user that a message was missed. Internally, each message can be tagged with a sequence number. If JMS detects that a message has been lost, it can report that there has been a sequence gap.

10 **[0042]** The multicast protocol also does not guarantee ordering. If messages are received out of order, this can be detected and reported as a sequence gap. Sequence gaps are reported by delivering an exception such as a SequenceGapException to an exception listener, such as ExceptionListener. If the session does not have an exception listener, 15 such as may be set using an setExceptionListener() method, no sequence gaps may be reported.

20 **[0043]** In one type of sequence gap, there is no way to get the message back. In a second type of sequence gap, the user may be receiving a long stream of information, such that a packet of information in the middle of a message can be missed, resulting in a corrupted message. In one embodiment in accordance with the present invention, the first packet of such a message contains information relating to the message, such as the origin of the message, the length of the message, a message

id, header properties, etc. The initial packet can be called upon to report some or all of this information to the user.

[0044] Every incoming message can also be tagged with information such that it is possible to know from which data stream the message is coming. Messages from different streams are not really competing with each other such that they would kill off one another when the traffic gets heavy. There can, however, be physical problems with the ethernet. The server may keep a buffer for recent messages, so the server can request them if they are missed.

[0045] For example, a system might use a routine or method to preserve a message stream by dropping a newer message, such as a KeepOld method. This may be desirable for applications that are more efficient in handling messages in a single stream. A system might also use a routine or method such as KeepNew, which can preserve a newer message by dropping an older message. This may be desirable for applications that are more efficient in handling messages that are repeated, but contain more timely information.

[0046] Depending on the amount of processing required for each multicast message, a client could easily fall behind. In this scenario, messages can start to pile up on the client. A new attribute, such as MessagesMaximum, can be added to limit the number of unprocessed messages that a given session can have outstanding. For a multicast session, messages can be thrown away once the client reaches the a

maximum message threshold. Valid values for a message maximum can be, for example, -1 and/or in the range of 1 to $2^{31}-1$. A value of -1 can indicate that there is no limit on the number of unprocessed messages that can accumulate on the session.

5 **[0047]** When messages are thrown away as a result of exceeding the maximum message value, they can be thrown away according to defined criteria, such as may be contained in a method or parameter, such as an overrun policy or OverrunPolicy. Valid values for OverrunPolicy can be, for example, 'KeepOld' and 'KeepNew'. Both the message maximum
10 and overrun policy can be configured via connection factories, and can be overridden on a per session basis using, for example, methods in a class such as weblogic.jms.extension.WLSession.

15 **[0048]** The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the
20 art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.